Adv. Appl. Clifford Algebras 27 (2017), 707–731

0188-7009/010707-25
published online September 27, 2016
DOI 10.1007/s00006-016-0722-6

Advances in Applied Clifford Algebras

Automatic Multivector Differentiation and Optimization

Lars Tingelstad* and Olav Egeland

Abstract. In this work, we present a novel approach to nonlinear optimization of multivectors in the Euclidean and conformal model of geometric algebra by introducing automatic differentiation. This is used to compute gradients and Jacobian matrices of multivector valued functions for use in nonlinear optimization where the emphasis is on the estimation of rigid body motions.

Keywords. Geometric Algebra, Automatic Differentiation, Optimization.

1. Introduction

Geometric algebra has been employed in many applications in robotics and computer vision. An important problem in these applications is the estimation of geometric objects and transformations from noisy data. Most estimation techniques based on geometric algebra employ singular value decomposition or other linear least squares methods, see [4,5,20,21,31,39]. Valkenburg and Alwesh [48] employ nonlinear optimization in a calibration method of multiple stationary 3D points as part of an optical positioning system using the conformal model of geometric algebra. Perwass [37] uses nonlinear optimization in 3D-reconstruction from multiple view geometry in the projective model of geometric algebra. The methods of [48] and [37] make use of multivector differentiation in geometric calculus [26] to compute gradients and Jacobian matrices. This involves tensor expressions.

Another tool for computing gradients and Jacobian matrices in nonlinear optimization is algorithmic or automatic differentiation [22]. Automatic differentiation computes derivatives with machine precision and works by exploiting the fact that all computer implementations of mathematical functions are composed of simple differentiable unary or binary operations, e.g., addition, multiplication or transcendental functions as sin, cos and exp.

*Corresponding author.

Derivatives of more complex functions are computed by applying the chain rule at each operation and bookkeeping the results [44].

Automatic differentiation has been used extensively in robotics and computer vision in recent years, e.g., for solving large scale bundle adjustment problems [2] and multi-camera calibration and people tracking in networks of RGB-D cameras [34]. These applications have been implemented in the Ceres [1] optimization framework developed by Google.

In this work, we present a novel approach to nonlinear optimization of multivectors in the Euclidean and conformal model of geometric algebra by introducing automatic differentiation. Automatic differentiation is used to compute gradients and Jacobian matrices of multivector valued functions for use in nonlinear optimization where the emphasis is on the estimation of rigid body motions.

The paper is organized as follows. Section 2 presents geometric algebra and the conformal model with focus on notation and on the representation of rigid body motions. Section 3 presents an overview of automatic differentiation. In Sect. 4 we introduce automatic multivector differentiation; A novel approach to differentiation of multivector valued functions by employing automatic differentiation. Further, in Sect. 5, we present multivector estimation using nonlinear optimization with focus on estimation of rigid body motions. In Sect. 6 we present implementation details regarding both automatic multivector differentiation and multivector estimation. Then, in Sect. 7, we present experimental results. Finally, in Sect. 8, we conclude the paper.

2. Geometric Algebra and the Conformal Model

Geometric algebra is an approach to geometry based on the work of W. Clifford which combined H. Grassmann's exterior algebra with Hamilton's quaternions and created what he termed geometric algebra. D. Hestenes developed geometric algebra further in his books [25,26] and later introduced the conformal model in [32].

The elements of a geometric algebra are called multivectors. The geometric algebra over 3-dimensional Euclidean space $\mathbb{R}^3$ is denoted $\mathbb{R}_3$. The notation $\mathbb{R}_3^r$ refers to the r-grade elements of $\mathbb{R}_3$ e.g. $\mathbb{R}_3^2$ refers to the elements of $\mathbb{R}_3$ of grade 2—the bivectors. The notation $\mathbb{R}_3^+$ refers to the elements of $\mathbb{R}_3$ of even grade. The conformal model of geometric algebra is denoted $\mathbb{R}_{4,1}$ and has the null basis $\{e_1, e_2, e_3, n_o, n_\infty\}$. The basis vector n_∞ represents the point at infinity, and the basis vector n_o represents an arbitrary origin. These basis vectors have the properties $n_\infty^2 = n_o^2 = 0$ and $n_\infty \cdot n_o = -1$. The notation e_{ij} is shorthand for the outer product $e_i \wedge e_j$ of the vectors $e_i, e_j \in \mathbb{R}_3^1$. The highest grade element of $\mathbb{R}_3$, the Euclidean pseudoscalar, is denoted I_3. The element of grade r of a multivector X is extracted using the grade projection operator $\langle X \rangle_r$. The reverse of a multivector X is denoted $\widetilde{X}$.

Vectors $x \in \mathbb{R}_3^1$ map to points $p \in \mathbb{R}_{4,1}^1$ using

$$p = x + \frac{1}{2}x^2 n_\infty + n_o. \tag{2.1}$$

A rotation in Euclidean space about a line through the origin is described by a rotor R, which can be written as the exponential

$$R = \cos\left(\frac{\theta}{2}\right) - \sin\left(\frac{\theta}{2}\right) B = \exp\left(-\frac{\theta}{2}B\right), \tag{2.2}$$

where $B \in \mathbb{R}^2_3$ is the unit bivector representing the rotation plane and θ is the rotation angle. It is noted that rotors are isomorphic to the unit quaternions. A rotor R is on the rotor manifold $\mathcal{R} \subset \mathbb{R}^+_3$. A rotor $R \in \mathcal{R}$ satisfies the constraint

$$R\widetilde{R} = 1 \tag{2.3}$$

where $\widetilde{R} = R^{-1}$ is the reverse rotor. The rotation of a vector $a \in \mathbb{R}^1_3$ to a vector a' is given by the sandwich product

$$a' = Ra\widetilde{R}. \tag{2.4}$$

A translation by a vector v is described with a translator T_v given by

$$T_v = 1 - \frac{vn_\infty}{2} = \exp\left(\frac{-vn_\infty}{2}\right). \tag{2.5}$$

The translation of the rotor R gives

$$T_v R\widetilde{T}_v = T_v \exp\left(-\frac{\theta}{2}B\right)\widetilde{T}_v = \exp\left(-\frac{\theta}{2}T_v B\widetilde{T}_v\right) \tag{2.6}$$

This gives

$$T_v R\widetilde{T}_v = \exp\left(-\frac{\theta}{2}(B - (v \cdot B)n_\infty)\right). \tag{2.7}$$

The position and orientation of rigid body can be described with a motor M. A motor can be described as a rotation about a line through the origin followed by a translation u according to

$$M = T_u R = \exp\left(\frac{-un_\infty}{2}\right)\exp\left(-\frac{\theta}{2}B\right) \tag{2.8}$$

This cannot be combined in a single exponential function because the rotation and the translation do not commute. This is solved with the application of Chasle's theorem, where the motion of a rigid body is described with a rotation about a line that is not necessarily through the origin, and a translation along the same line, in which case the rotation and translation will commute. The motor is then

$$M = T_w T_v R\widetilde{T}_v = \exp\left(-\frac{\theta}{2}(B - (v \cdot B)n_\infty) - \frac{wn_\infty}{2}\right) \tag{2.9}$$

where w is the translation along the line of rotation, and v is the translation of the rotor. This vector is in the rotation plane given by B. A motor can therefore be written as the exponential function

$$M = \exp\left(-\frac{1}{2}\Lambda^*\right), \tag{2.10}$$

where $\Lambda^* \in \text{span}\{e_{12}, e_{13}, e_{23}, e_1 n_\infty, e_2 n_\infty, e_3 n_\infty\}$ and

$$\Lambda^* = \theta B + t n_\infty.$$

Here $B \in \mathbb{R}^2_3$ is the rotation plane, and $t \in \mathbb{R}^1_3$ is a vector given by $t = t_\perp + t_\parallel$ where $t_\perp = w$ is normal to B, and $t_\parallel = -\theta(v \cdot B)$ is in the plane defined by B.

Moreover, it can be shown that Λ^* is a dual line representing the screw axis of the rigid body motion, see [12]. Following [50], the exponential formulation in (2.10) can be written in terms of the constituent elements of different grades as

$$\langle M \rangle_0 = \cos\left(\frac{\theta}{2}\right) \tag{2.11}$$

$$\langle M \rangle_2 = \sin\left(\frac{\theta}{2}\right) B + \cos\left(\frac{\theta}{2}\right) t_\perp n_\infty + \text{sinc}\left(\frac{\theta}{2}\right) t_\parallel n_\infty \tag{2.12}$$

$$\langle M \rangle_4 = \sin\left(\frac{\theta}{2}\right) B t_\perp n_\infty, \tag{2.13}$$

where $\text{sinc}(x) = \sin(x)/x$.

The motor manifold $\mathcal{M}$ is of dimension 6, and is embedded in the 8-dimensional subspace $\mathbb{M}$ with basis $\{1, e_{12}, e_{13}, e_{23}, e_1 n_\infty, e_2 n_\infty, e_3 n_\infty, I_3 n_\infty\}$. A multivector $M \in \mathbb{M}$ will be a motor if and only if

$$M\widetilde{M} = 1. \tag{2.14}$$

It is noted that this condition implies

$$\langle MM \rangle_4 = 0. \tag{2.15}$$

A point $p \in \mathbb{R}^1_{4,1}$ is displaced by a motor $M \in \mathcal{M}$ using

$$p' = Mp\widetilde{M}, \tag{2.16}$$

where p' is the displaced point.

3. Automatic Differentiation

Automatic differentiation computes derivative values of computer implemented functions with accuracy up to machine precision. Derivative information as Jacobian matrices and gradients can be found symbolically by hand-computations and implemented manually or generated by computer algebra systems, approximated using numerical differentiation or computed using automatic differentiation. Supplying hand-coded derivative values is error-prone and time consuming for complex nonlinear functions and symbolic differentiation using a computer algebra system can in certain cases lead to significant long computation times [28,44]. Consider the vector valued function $f : \mathbb{R}^n \to \mathbb{R}^m$,

$$\boldsymbol{y} = f(\boldsymbol{x}) \tag{3.1}$$

with directional derivative

$$\dot{\boldsymbol{y}} = f'(\boldsymbol{x})\dot{\boldsymbol{x}} = \lim_{h\to 0} \frac{f(\boldsymbol{x}+h\dot{\boldsymbol{x}}) - f(\boldsymbol{x})}{h}, \ h \in \mathbb{R}. \tag{3.2}$$

Numerical differentiation using finite differences is based on evaluating (3.2) with a small value $h > 0$. However, this method is prone to truncation and rounding off errors and may fail completely when the implemented functions include conditional statements [44]. Automatic differentiation is also numerical differentiation in that it computes numerical derivative values, but it computes the numerical values up to machine precision.

The main principle behind automatic differentiation is that every function can be represented as a finite sequence φ of elemental unary or binary operations with known derivatives, e.g., unary operations as trigonometric, exponential and logarithmic operations or binary operations as addition, multiplication, division and the power operation. The derivative of the whole computation can then be computed through the chain rule. Using the notation of [22], a finite sequence can be written as

$$\varphi = \{v_{-n}, \ldots, v_0, v_1, \ldots, v_{l-m}, v_{l-m+1}, \ldots, v_l\}, \tag{3.3}$$

where $\{v_{-n}, \ldots, v_0\}$ are the input variables, and $\{v_1, \ldots, v_{l-m}\}$ are intermediate variables computed as

$$v_i = \varphi_i(v_0, \ldots, v_{i-1}), \ i \in \{0, \ldots, l-m\}. \tag{3.4}$$

The output of the sequence is given by the variables $\{v_{l-m+1}, \ldots, v_l\}$. A finite sequence for the computations of a computer implemented function can be determined by what is known as an evaluation trace of elemental operations or Wengert list [3]. As an example consider the function $f : \mathbb{R}^2 \to \mathbb{R}$,

$$y = f(x_1, x_2) = x_2 \sin(x_1^2) \tag{3.5}$$

with the evaluation trace as shown in Table 1.

There are two main approaches to automatic differentiation – forward mode and reverse mode.

TABLE 1. Forward evaluation and derivative trace of the function $y = f(x_1, x_2) = x_2 \sin(x_1^2)$

Forward evaluation trace		Forward derivative trace	
v_{-1}	$= x_1$	$\dot{v}_{-1}$	$= \dot{x}_1$
v_0	$= x_2$	$\dot{v}_0$	$= \dot{x}_2$
v_1	$= v_{-1} \times v_{-1}$	$\dot{v}_1$	$= \dot{v}_{-1} \times v_{-1} + v_{-1}\dot{v}_{-1}$
v_2	$= \sin v_1$	$\dot{v}_2$	$= \dot{v}_1 \cos v_1$
v_3	$= v_0 \times v_2$	$\dot{v}_3$	$= \dot{v}_0 \times v_2 + v_0 \times \dot{v}_2$
y	$= v_3$		

3.1. Forward Mode

Forward mode automatic differentiation is based on assigning intermediate variable v_i a derivative $\dot{v}_i$ with respect to input variable j

$$\dot{v}_i = \frac{\partial v_i}{\partial x_j}. \tag{3.6}$$

The forward derivative trace is then found by applying the chain rule to each elemental operation in the forward evaluation trace. For a function $f : \mathbb{R}^n \to \mathbb{R}$, one pass through the forward derivative trace computes $\partial f/\partial x_j$ for a fixed j. The Jacobian matrix of a function $f : \mathbb{R}^n \to \mathbb{R}^m$ can thus be evaluated in n passes through the forward derivative trace, where each pass computes one column.

Consider again the function in (3.5). The derivative $\partial f/\partial x_1 = \partial v_3/\partial v_{-1}$ can be found by setting $\dot{v}_{-1} = 1$ and $\dot{v}_0 = 0$ and evaluating the forward derivative trace in Table 1:

$$\frac{\partial v_3}{\partial v_{-1}} = 2v_{-1}v_0 \cos v_1 = 2x_1x_2 \cos x_1^2. \tag{3.7}$$

Similarly, $\partial f/\partial x_2$ can be found by setting $\dot{v}_{-1} = 0$ and $\dot{v}_0 = 1$ and evaluating the forward derivative trace again.

3.1.1. Dual Numbers. Forward mode automatic differentiation can be seen as evaluating a function f using dual numbers. Introduced by W. K. Clifford in the seminal paper *Preliminary Sketch of Biquaternions* [6], dual numbers are given as

$$z = x + \varepsilon y \tag{3.8}$$

where x and y are real numbers, and ε is the dual unit, which satisfies $\varepsilon \neq 0$ and $\varepsilon^2 = 0$. The Taylor expansion of $f(x + \varepsilon)$ at x is given by

$$f(x + \varepsilon) = f(x) + \varepsilon f'(x) \tag{3.9}$$

which returns the function value as the real part plus the derivative as the dual part.

3.2. Reverse Mode

Reverse mode automatic differentiation is based on populating the intermediate variables in the forward evaluation trace and then propagating derivatives with respect to an output variable y_j in a reverse phase. This is done by assigning to intermediate variable v_i the adjoint

$$\bar{v}_i = \frac{\partial y_j}{\partial v_i}, \tag{3.10}$$

and propagating these backwards from a given output. The reverse adjoint trace of (3.5) is shown in Table 2. As seen, both $\bar{v}_0$ and $\bar{v}_{-1}$ is computed in one reverse pass. This is the major advantage of the reverse mode compared to the forward mode of automatic differentiation, that is, the gradient of a function $f : \mathbb{R}^n \to \mathbb{R}$ can be evaluated in one pass compared to n passes in the forward mode.

TABLE 2. Forward evaluation trace and reverse adjoint trace of the function $y = f(x_1, x_2) = x_2 \sin(x_1^2)$

Forward evaluation trace		Reverse adjoint trace	
v_{-1}	$= x_1$	$\bar{v}_{-1}$	$= \bar{x}_1$
v_0	$= x_2$	$\bar{v}_0$	$= \bar{x}_2$
v_1	$= v_{-1} \times v_{-1}$	$\bar{v}_{-1}$	$= \bar{v}_1 \frac{\partial v_1}{\partial v_{-1}} = v_0 \cos v_1 2v_{-1}$
v_2	$= \sin v_1$	$\bar{v}_1$	$= \bar{v}_2 \frac{\partial v_2}{\partial v_1} = v_0 \cos v_1$
v_3	$= v_0 \times v_2$	$\bar{v}_2$	$= \bar{v}_3 \frac{\partial v_3}{\partial v_2} = 1 \times v_0$
		$\bar{v}_0$	$= \bar{v}_3 \frac{\partial v_3}{\partial v_0} = 1 \times v_2$
y	$= v_3$	$\bar{v}_3$	$= 1$

3.3. Implementation Using Operator Overloading

There are two main approaches to implementing automatic differentiation—source code transformation and operator overloading. Source code transformation is based on pre-processing the function source code and generating code that implements the necessary steps to compute the derivatives. The approach used in this work is that of operator overloading in C++. The main idea here is to overload the scalar type used in the computations and to write functions as function templates using template metaprogramming. The new scalar type then implements the necessary logic to compute the derivatives. Examples are the `Jet`-type in the Google Ceres framework that implements forward mode automatic differentiation using the dual numbers approach in Sect. 3.1.1 and the `adouble`-type in the Adept [29] framework by Robin J. Hogan that implements both forward and reverse mode automatic differentiation using expression templates [30].

4. Automatic Multivector Differentiation

As an example, consider differentiating the sandwich product

$$f = Ra\widetilde{R} \tag{4.1}$$

where $R = \exp(B) \in \mathcal{R} \subset \mathbb{R}_3^+$, $B = -\frac{\theta}{2} e_{12} \in \mathbb{R}_3^2$, $a = e_1 \in \mathbb{R}_3^1$ with respect to θ. Evaluating f analytically, and using $\varphi = \frac{\theta}{2}$ gives

$$\begin{aligned}
f(\theta) &= (\cos(\varphi) - \sin(\varphi)e_{12})\, e_1 (\cos(\varphi) + \sin(\varphi)e_{12}) && (4.2)\\
&= (\cos(\varphi)e_1 - \sin(\varphi)e_{12}e_1)(\cos(\varphi) + \sin(\varphi)e_{12}) && (4.3)\\
&= (\cos(\varphi)e_1 + \sin(\varphi)e_2)(\cos(\varphi) + \sin(\varphi)e_{12}) && (4.4)\\
&= \cos^2(\varphi)e_1 + \sin(\varphi)\cos(\varphi)e_1 e_{12} && (4.5)\\
&\quad + \sin(\varphi)\cos(\varphi)e_2 + \sin^2(\varphi)e_2 e_{12} && (4.6)\\
&= (\cos^2(\varphi) - \sin^2(\varphi))e_1 + 2\sin(\varphi)\cos(\varphi)e_2 && (4.7)\\
&= \cos(\theta)e_1 + \sin(\theta)e_2, && (4.8)
\end{aligned}$$

with derivative

$$\frac{df}{d\theta} = -\sin(\theta)e_1 + \cos(\theta)e_2. \tag{4.9}$$

TABLE 3. Evaluation trace of the sandwich product $f = Ra\widetilde{R}$ where $R = \exp(B) \in \mathcal{R} \subset \mathbb{R}_3^+$, $B = -\frac{\theta}{2}e_{12} \in \mathbb{R}_3^2$, $a = e_1 \in \mathbb{R}_3^1$ with respect to θ

Forward evaluation trace		Forward derivative trace	
v_{-1}	$= e_1$	$\dot{v}_{-1}$	$= \dot{e}_1 = 0$
v_0	$= \theta$	$\dot{v}_0$	$= \dot{\theta} = 1$
v_1	$= 0.5 \times v_0$	$\dot{v}_1$	$= 0.5 \times \dot{v}_0$
v_2	$= \cos(v_1)$	$\dot{v}_2$	$= -\sin(v_1)\dot{v}_1$
v_3	$= \sin(v_1)$	$\dot{v}_3$	$= \cos(v_1)\dot{v}_1$
v_4	$= -1 \times v_3$	$\dot{v}_4$	$= -1 \times \dot{v}_3$
v_5	$= v_2 \times v_{-1}$	$\dot{v}_5$	$= \dot{v}_2 \times v_{-1} + v_2 \times \dot{v}_{-1}$
v_6	$= v_4 \times v_{-1}$	$\dot{v}_6$	$= \dot{v}_4 \times v_{-1} + v_4 \times \dot{v}_{-1}$
v_7	$= -1 \times v_6$	$\dot{v}_7$	$= -1 \times \dot{v}_6$
v_8	$= v_5 \times v_2$	$\dot{v}_8$	$= \dot{v}_5 \times v_2 + v_5 \times \dot{v}_2$
v_9	$= v_5 \times v_3$	$\dot{v}_9$	$= \dot{v}_5 \times v_3 + v_5 \times \dot{v}_3$
v_{10}	$= 1 \times v_9$	$\dot{v}_{10}$	$= -1 \times \dot{v}_9$
v_{11}	$= v_7 \times v_2$	$\dot{v}_{11}$	$= \dot{v}_7 \times v_2 + v_7 \times \dot{v}_2$
v_{12}	$= v_7 \times v_3$	$\dot{v}_{12}$	$= \dot{v}_7 \times v_3 + v_7 \times \dot{v}_3$
v_{13}	$= -1 \times v_{12}$	$\dot{v}_{13}$	$= -1 \times \dot{v}_{12}$
v_{14}	$= v_8 + v_{13}$	$\dot{v}_{14}$	$= \dot{v}_8 + \dot{v}_{13}$
v_{15}	$= v_{10} + v_{11}$	$\dot{v}_{15}$	$= \dot{v}_{10} + \dot{v}_{11}$

The finite sequence or evaluation trace of (4.2) to (4.8) are shown in Table 3 and consist of a total of 17 statements to compute the derivative with respect to θ. The multiplications in statements v_7, v_{10} and v_{13} correspond to the sign changes due to the geometric products in (4.2) and (4.6).

The expressions in statements v_{14} and v_{15} are the output of the function f, i.e.,

$$v_{14} = f_1 = \cos(\theta)e_1 \tag{4.10}$$
$$v_{15} = f_2 = \sin(\theta)e_2, \tag{4.11}$$

whereas the statements $\dot{v}_{14}$ and $\dot{v}_{15}$ are the corresponding derivatives

$$\dot{v}_{14} = \frac{\partial f_1}{\partial \theta} = -\sin(\theta)e_1 \tag{4.12}$$
$$\dot{v}_{15} = \frac{\partial f_2}{\partial \theta} = \cos(\theta)e_2. \tag{4.13}$$

Comparing (4.10) and (4.11) with (4.8) and comparing (4.12) and (4.13) with (4.9) show that the output is correct and that computer implementations of geometric algebra algorithms can be differentiated using automatic differentiation.

5. Multivector Estimation

In this section a new approach to rotor and motor estimation is proposed based on nonlinear least squares optimization of the form

$$\min_{X\in\mathcal{M}} F(X). \tag{5.1}$$

where $F : \mathcal{M} \mapsto \mathbb{R}$ is the cost function to be minimized and X is a conformal motor on the motor manifold $\mathcal{M}$ or a Euclidean rotor on the rotor manifold $\mathcal{R}$.

The cost function F is given by

$$F(X) = \frac{1}{2}\sum_{i=1}^{N}(f_i(X))^2 = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{p}(r_{ij}(X))^2, \tag{5.2}$$

where each observation i corresponds to a vector $f_i(X) = (r_{i1}, \ldots, r_{ip})^\top$ of p residuals $r_{ij} \in \mathbb{R}$. It is seen that the cost function F has $m = Np$ residuals where N is the number of observations. A residual is a scalar measure for the discrepancy between the model and the observed data [36].

5.1. Rotor Estimation

Following Lasenby et al. [31], the rotor estimation problem can be formulated as the nonlinear least squares optimization problem

$$\min_{R\in\mathcal{R}} \frac{1}{2}\sum_{i=1}^{N}\left(Ra_i\widetilde{R} - b_i\right)^2, \tag{5.3}$$

where $a_i, b_i \in \mathbb{R}^1_3$ and $\{(a_i, b_i)\}_{i=1,\ldots,N}$ are N observations of Euclidean vectors and the rotor R is parameterized by the parameter vector $\boldsymbol{x}$. In this formulation there are 3 residuals for each observation, namely, the residual Euclidean distances in the e_1, e_2 and e_3 directions, giving a total of $3N$ residuals. The Jacobian matrices will be of size $3N \times \dim(\boldsymbol{x})$.

5.1.1. Parameterization. Several parameterizations are possible for the rotor in (5.3) such that the constraint in (2.3) is enforced. One possible parameterization is to use all the 4 components of the rotor

$$R(\boldsymbol{x}) = \sum_{i=1}^{4} x_i R_i,\ \ \boldsymbol{x} \in \mathbb{R}^4,\ \ R_i \in \{1, e_{12}, e_{13}, e_{23}\}, \tag{5.4}$$

and to re-normalize after each optimization step. However, this is an over-parameterization as the rotors lie on the 3-dimensional manifold $\mathcal{R}$.

Another solution to the problem of over-parameterization is to parameterize the rotor using only the bivector parts (x_2, x_3, x_4) of $\boldsymbol{x}$ and to compute the full parameterization of the rotor using

$$\boldsymbol{x} = (\sqrt{1 - x_2^2 - x_3^2 - x_4^2}, x_2, x_3, x_4). \tag{5.5}$$

This is investigated for quaternions in [40]. This approach is not recommended as the rotation angle is limited to $\langle -\frac{\pi}{2}, \frac{\pi}{2}\rangle$ and the radicand in (5.5) can not be guaranteed to be positive when estimating the rotor R using the Levenberg–Marquardt method.

The parameterization of the rotor R in this work makes use of the theory of optimization on differentiable manifolds [44]. Then the step is calculated in the tangent space of the rotor manifold $\mathcal{R}$ and thus removing null directions in the step update. The increment is calculated in the rotor manifold using the exponential map, which is given in terms of its bivector generator B:

$$R = \exp(B(\boldsymbol{x})), \tag{5.6}$$

where

$$B(\boldsymbol{x}) = \sum_{i=1}^{3} x_i B_i, \;\; \boldsymbol{x} \in \mathbb{R}^3, \;\; B_i \in \{e_{12}, e_{13}, e_{23}\}, \tag{5.7}$$

The rotor in iteration $k+1$ can then be written as

$$R_{k+1} = \exp(B(\boldsymbol{x}))R_k, \tag{5.8}$$

where the exponential map is computed using (2.2). It follows that $R_{k+1} \in \mathcal{R}$ whenever $R_k \in \mathcal{R}$.

5.2. Motor Estimation from Point Clouds

Consider a rigid body that is displaced by a motor $M \in \mathcal{M}$. Let $\{p_i\}, p_i \in \mathbb{R}^1_{4,1}$ be a set of points on the rigid body in the initial configuration, and let $\{q_i = Mp_i\widetilde{M}\}$ be the same point in the displaced configuration. The sets $\{p_i\}$ and $\{q_i\}$ are called point clouds. Motor estimation is the problem of finding the motor M given $\{p_i\}$ and $\{q_i\}$.

One possible formulation of this optimization problem is to use the inner product between two conformal points

$$\min_{M \in \mathcal{M}} \frac{1}{2} \sum_{i=1}^{N} \left(Mp_i\widetilde{M} \cdot q_i \right)^2. \tag{5.9}$$

In this formulation the measure that is optimized is the squared distance between each two points, resulting in a 1-dimensional residual vector for each observation. This, however, is not a good formulation for the cost function as discussed in the experimental results in Sect. 7.2.2.

A better formulation of this problem is to project the points to the 3-dimensional Euclidean model after the transformation by the motor M, and then to use the residual errors along each of the coordinate axes, resulting in a 3-dimensional residual vector for each observation with the optimization problem

$$\min_{M \in \mathcal{M}} \frac{1}{2} \sum_{i=1}^{N} \left(\mathrm{P}_{\mathbb{R}^1_3}(Mp_i\widetilde{M}) - \mathrm{P}_{\mathbb{R}^1_3}(q_i) \right)^2. \tag{5.10}$$

Implementationwise, this projection is performed by selecting only the pure Euclidean components of the conformal points.

5.2.1. Parameterization. One possible parameterization of conformal motors is based on the polar decomposition by Valkenburg and Dorst [49]. This parameterization is based on representing the motor M using the full 8-dimensional basis as presented in Sect. 2, that is,

$$M(\boldsymbol{x}) = \sum_{i=1}^{8} x_i M_i, \tag{5.11}$$

where $\boldsymbol{x} = (x_1, \ldots, x_8)^\top$ and $M_i \in \{1, e_{12}, e_{13}, e_{23}, e_1 n_\infty, e_2 n_\infty, e_3 n_\infty, I_3 n_\infty\}$. Using this approach, similarly to the 4-dimensional rotor parameterization in Sect. 5.1.1, it is necessary to re-normalize after each optimization step to ensure that the resulting object $X(\boldsymbol{x}) \in \mathbb{M}$ is in fact a motor, in which case

$$\langle X(\boldsymbol{x}) \widetilde{X}(\boldsymbol{x}) \rangle_4 = 0. \tag{5.12}$$

This re-normalization can not be performed by normalizing $\boldsymbol{x}$ due to the constraint in (5.12). Lemma 2.3 in [49], however, shows that any element $X \in \mathbb{M}$, $|X| \neq 0$ has a unique polar decomposition $X = MS = SM$ where $M \in \mathcal{M}$, $S \in \operatorname{span}\{1, I_3 n_\infty\}$, $\langle S \rangle > 0$. Any element $X \in \mathbb{M}$ can then be projected onto the motor manifold $\mathcal{M}$ using the following projection

$$\mathrm{P}_{\mathcal{M}}(X) = XS^{-1} = \frac{X}{|X|}\left(1 - \frac{\langle X\widetilde{X}\rangle_4}{2\langle X\widetilde{X}\rangle}\right). \tag{5.13}$$

The motor in iteration $k+1$ using this parameterization is then computed as

$$M_{k+1} = \mathrm{P}_{\mathcal{M}}\left(M_k + X(\boldsymbol{x})\right). \tag{5.14}$$

Another parameterization is based on the exponential form of a motor in terms of its bivector generator as presented in (2.10). As opposed to the polar decomposition approach, this ensures that the step is taken in the motor manifold $\mathcal{M}$. The motor in iteration $k+1$ using this parameterization is given by

$$M_{k+1} = \exp(\Lambda^*(\boldsymbol{x}))M_k, \tag{5.15}$$

where

$$\Lambda^*(\boldsymbol{x}) = \sum_{i=1}^{6} x_i \Lambda_i^*,\ \boldsymbol{x} \in \mathbb{R}^6,\ \Lambda_i^* \in \{e_{12}, e_{13}, e_{23}, e_1 n_\infty, e_2 n_\infty, e_3 n_\infty\}, \tag{5.16}$$

and the closed form of $\exp(\Lambda^*(\boldsymbol{x}))$ is given in (2.11)–(2.13). Then $M_{k+1} \in \mathcal{M}$ whenever $M_k \in \mathcal{M}$.

6. Implementation

This section presents the implementation of automatic multivector differentiation and estimation.

LISTING 1. Templated C++ code of the sandwich product $b = Ra\widetilde{R}$ where $a, b, \in \mathbb{R}^1_3$ and $R \in \mathbb{R}^+_3$ for use in automatic differentiation

```
template <typename T>
void RotorRotateVector(const T r[4], const T v[3], T result[3]) {
  Rotor<T> rotor{r[0], r[1], r[2], r[4]};  // {1, e12, e13, e23}
  Vector<T> a{v[0], v[1], v[2]};           // {e1, e2, e3}
  Vector<T> b = rotor * a * rotor.rev();   // {e1, e2, e3}
  for (int i = 0; i < 3; ++i) result[i] = b[i];
}
```

6.1. Automatic Multivector Differentiation

The use of automatic differentiation using operator overloading sets some constraints on the implementation of geometric algebra and the conformal model. A number of geometric algebra libraries and software systems in different programming languages have been developed over the last decades, e.g., CLUCalc [38] and GAViewer [15] which are both designed as development environments with graphics for visualization based on OpenGL [51] and specially designed domain specific languages [19] for ease of programming. Examples of freely available geometric algebra C++ libraries are Gaalop [27], which is a geometric algebra pre-compiler for high-performance computing where algorithms developed in CLUCalc can be used directly in the C++ source and pre-compiled to, e.g., C++, OpenCL [45] and CUDA [35]. Other geometric algebra source code generators are Gaigen [18] and Gaigen2 [16] which are used in the C++ code accompanying the Geometric Algebra for Computer Science book by Dorst et al. [14]. In the Gaigen libraries C, C++ and Java code can be generated from specifications of the Euclidean, projective and conformal model of geometric algebra. However, the aforementioned libraries are not suited for use with operator overloading based automatic differentiation libraries as they do not permit templating the scalar type used in the computations.

In this paper we propose an approach to multivector estimation using automatic differentiation based on the following idea: Consider again the sandwich product in (4.1). The main idea is to be able to write code as presented in Listing 1 where the input are the four components of a rotor $R \in \mathbb{R}^+_3$ and the three components of a Euclidean vector $v \in \mathbb{R}^1_3$ and the output is the three components of the rotated vector. In addition to the geometric product and reverse operation in the Euclidean model presented in Listing 1, more complex operations like the outer and inner products and the left contraction should be supported as well as the conformal model and other geometric algebras. To clarify this, we present three possible formulations to investigate their suitability to our problem. The three formulations are a matrix based implementation of the Euclidean model and two multivector based implementations of both the Euclidean and conformal model implemented in the C++11 standard.

LISTING 2. Templated C++ code of the sandwich product $b = Ra\widetilde{R}$ where $a, b, \in \mathbb{R}^1_3$ and $R \in \mathbb{R}^+_3$ for use in automatic differentiation implemented using 4×4 matrices in the Eigen [23] matrix library

```
template <typename T>
void RotorRotateVector(const T r[4], const T v[3], T result[3]) {
  Eigen::Matrix<T, 4, 4> rotor = r[0] * id<T>() + r[1] * e1<T>() * e2<T>() +
                                 r[2] * e1<T>() * e3<T>() +
                                 r[3] * e2<T>() * e3<T>();
  Eigen::Matrix<T, 4, 4> a = v[0] * e1<T>() + v[1] * e2<T>() + v[2] * e3<T>();
  Eigen::Matrix<T, 4, 4> b = rotor * a * rotor.inverse();
  result[0] = b(0, 3);  // e1
  result[1] = b(0, 2);  // e2
  result[2] = b(0, 0);  // e3
}
```

LISTING 3. Templated matrix implementation of the basis vector $e_1 \in \mathbb{R}^1_3$. The other basis vectors e_2, e_3 are implemented similarly

```
template <typename T>
Eigen::Matrix<T, 4, 4> e1() {
  Eigen::Matrix<T, 4, 4> m;
  m << T(0), T(0), T(0), T(1), T(0), T(0), T(1), T(0), T(0), T(1), T(0), T(0),
       T(1), T(0), T(0), T(0);
  return m;
}
```

6.1.1. Matrix Based Implementation of Geometric Algebra. A matrix implementation of the Euclidean model can be based on the isomorphism between the geometric algebra $\mathbb{R}_3$ and the matrix algebra of 4×4 matrices with entries in $\mathbb{R}$, see [42]. The three basis vectors in $\mathbb{R}^1_3$ can then be represented by the matrices

$$e_1 \simeq \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}, e_2 \simeq \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{pmatrix}, e_3 \simeq \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}. \tag{6.1}$$

Then the geometric product is matrix multiplication and the identity is represented by the 4×4 identity matrix. Using the Eigen [23] C++ matrix library the function in Listing 1 can be implemented as presented in Listing 2 and Listing 3. Further, the conformal model can be implemented using Vahlen matrices, see [13] and [43].

This approach is relatively simple to implement for the Euclidean model as all elements in the geometric algebra are represented by matrices. However, and as noted in [14], the matrix representation works only for the geometric product and the contraction operations can not be implemented in the same framework due to nonassociativity. The simplicity of representing all elements in a geometric algebra as matrices is also the main argument for not using it as it lacks the notion of types, that is, a vector has the same type as a bivector or a rotor. This kind of abstraction is essential for working with all the geometric entities and transformations in, e.g., the conformal model.

LISTING 4. Templated C++ code of the sandwich product $b = Ra\widetilde{R}$ where $a, b, \in \mathbb{R}^1_3$ and $R \in \mathbb{R}^+_3$ for use in automatic differentiation implemented using the hep-ga [41] library

```
template <typename T>
void RotorRotateVector(const T r[4], const T v[3], T result[3]) {
  using EGA = hep::algebra<T, 3, 0>;
  using Vector = hep::multi_vector<EGA, hep::list<1, 2, 4>>;
  using Rotor = hep::multi_vector<EGA, hep::list<0, 3, 5, 6>>;
  Rotor rotor{r[0], r[1], r[2], r[3]};           // {1, e12, e13, e23}
  Vector a{v[0], v[1], v[2]};                    // {e1, e2, e3}
  Vector b = hep::grade<1>(rotor * a * ~rotor);  // {e1, e2, e3}
  for (int i = 0; i < 3; ++i) result[i] = b[i];
}
```

LISTING 5. Type alias for a conformal point $p \in \mathbb{R}^1_{4,1}$ in the hep-ga [41] library

```
template <typename T>
using CGA = hep::algebra<T, 4, 1>;
template <typename T>
using Point = hep::multi_vector<CGA, hep::list<1, 2, 4, 8, 16>>;
```

6.1.2. Multivector Based Implementations of Geometric Algebra. This section presents the two multivector based implementations of geometric algebra.

The first implementation is the hep-ga [41] library developed by Chris Schwan for use in high energy physics. hep-ga is a C++11 library for efficient numeric computations using geometric algebra and is implemented using C++ expression templates [30]. The implementation of the geometric algebra computations in hep-ga follows the bitset approach by Daniel Fontijne in [17] and [14]. The use of expression templates ensures high performance computations at runtime, however the compile times using this library can become very long especially when evaluating multiple sandwich products of multivectors with many components, e.g., in forward kinematic computations of serial kinematic chains using motors.

The implementation of the function in Listing 1 using the hep-ga library is shown in Listing 4. Using this approach, different multivectors have different types through the use of type aliases. Similarly, the conformal model can be implemented by defining special types for the algebra and for the different multivectors. Only non-degenerate diagonal metrics are implemented and the change to a non-diagonal null metric must be implemented by the user and computed at runtime. The type alias for a conformal point $p \in \mathbb{R}^1_{4,1}$ is shown in Listing 5.

The second implementation of geometric algebra used in this work is the Versor [7] library developed by Pablo Colapinto. Versor is a C++11 template metaprogramming library that generates optimized code at compile time, however, in contrast to the hep-ga library it is not based on expression templates. The Versor library supports arbitrary dimensions and non-degenerate metrics and implements the change to a null basis in the conformal model at

LISTING 6. Templated C++ code of the sandwich product $b = Ma\widetilde{M}$ where $a, b, \in \mathbb{R}^1_{4,1}$ and $M \in \mathcal{M}$ for use in automatic differentiation implemented using the Versor library

```
template <typename T>
void MotorRotatePoint(const T m[8], const T p[5], T result[5]) {
  Motor<T> motor{m[0], m[1], m[2], m[3],     // {1, e12, e13, e23,
                 m[4], m[5], m[6], m[7]};    //  e1ni, e2ni, e3ni, e123ni}
  Point<T> a{p[0], p[1], p[2], p[3], p[4]};  // {e1, e2, e3, no, ni}
  Point<T> b = a.spin(motor);                // b = M a ~M
  for (int i = 0; i < 5; ++i) result[i] = b[i];
}
```

compile time. This is vital for high runtime performance and for algorithm development. Similarly to the hep-ga library, the implementation of the geometric algebra computations in Versor is following the bitset approach by Daniel Fontijne in [17] and [14]. The Versor library also includes implementation of algorithms like the logarithm and exponential maps of conformal motors and data types for the most used conformal objects, e.g., vectors, points, spheres, lines and planes as well as rotors and motors. Examples of the use of the Versor library for surface and mesh generation using the conformal model can be found in [8] and [9].

The implementation of the function in Listing 1 using the Versor library is similar to the code for the hep-ga library in the sense that type aliases are used for the different multivectors. However, the types for the conformal model can be defined in a null metric, that is, a point $p \in \mathbb{R}^1_{4,1}$ can be defined on the basis $\{e_1, e_2, e_3, n_o, n_\infty\}$ directly. The change to the diagonal metric and back is performed at compile time. This enables us to write code as presented in Listing 6 and compute a 5×8 Jacobian matrix of the sandwich product $b = Ma\widetilde{M}$ where $M \in \mathcal{M}$ and $a, b \in \mathbb{R}^1_{4,1}$.

6.2. Multivector Estimation

The optimization framework used in this work is the Ceres [1] nonlinear least squares optimization framework developed by Google for solving large scale bundle adjustment problems [2,24]. The Ceres framework is written in C++ and supports multiple line search and trust region methods, including nonlinear conjugate gradients, BFGS, L-BFGS, Powell's Dogleg trust region method and the Levenberg-Marquardt method [33]. The Ceres framework also supports multiple sparse and dense linear solvers through the Eigen [23] and SuiteSparse [11] libraries. Gradients and Jacobians can be supplied manually or evaluated using numerical finite difference methods and automatic differentiation using an implementation of dual number forward mode automatic differentiation, as presented in Sect. 3.1.1, through the `Jet` class. Another important feature of the Ceres framework is the use of OpenMP [10] based multithreading to speed up Jacobian evaluations and linear solvers.

There are two main aspects of implementing optimization problem using automatic differentiation in the Ceres framework. The first is to implement a templated functor (function object) that implements the logic of the cost function and to pass this functor to the optimization problem using the

LISTING 7. Templated functor for use in the Ceres framework that implements the cost function in (5.10) using the Versor library

```
struct EstimateMotor3ResidualsCostFunctor {
  EstimateMotor3ResidualsCostFunctor(const Point<double>& a,
                                     const Point<double>& b)
      : a_(a), b_(b) {}

  template <typename T>
  bool operator()(const T* const m, T* residual) const {
    Motor<T> motor{m[0], m[1], m[2], m[3],   // {1, e12, e13, e23,
                   m[4], m[5], m[6], m[7]};  //  e1ni, e2ni, e3ni, e123ni}
    Point<T> a{a_};
    Point<T> b{b_};
    Point<T> c = a.spin(motor);
    for (int i = 0; i < 3; ++i) residual[i] = c[i] - b[i];
    return true;
  }

 private:
  const Point<double> a_;
  const Point<double> b_;
};
```

`AutoDiffCostFunction` class. The second aspect is used if the parameter to be estimated is an over-parameterization, as in rotor and motor estimation where the parameters are subject to the constraint of being on the rotor and motor manifolds. This is performed by implementing a templated functor that implements, e.g., the exponential map of rotors and motors and passing this to the optimization problem using the `AutoDiffLocalParameterization` class.

The functor that implements the cost function in (5.10) using the Versor library is shown in Listing 7 and the implementation of the motor parameterization in (2.10) is shown in Listing 8.

7. Experimental Results

This section presents the experimental results of automatic multivector differentiation and estimation using synthetic datasets.

The experiments were implemented using our framework called GAME [46] for multivector estimation. The source code is available online. The GAME framework is implemented in an Ubuntu[1] 16.04 Docker[2] container. All experiments were run on an early 2015 Apple Macbook Pro 13 with an Intel i5 CPU at 2.7 GHz with 8 GB memory. The Docker version used were v1.12.0. The Ceres version were v.1.11.0 with Eigen v.3.2.92. The Adept version used were v.1.1.

7.1. Automatic Differentiation of Multivector Valued Functions

This section presents experimental results from calculation of the 3×1 Jacobian matrix of the sandwich product presented in (4.2)–(4.8) and evaluated at

[1] http://www.ubuntu.com/.
[2] https://www.docker.com.

LISTING 8. Templated functor for use in the Ceres framework that implements the motor parameterization in (2.10) using the Versor library

```
struct MotorExponentialMapLocalParameterization {
  template <typename T>
  bool operator()(const T* x, const T* delta, T* x_plus_delta) const {
    Motor<T> motor_1;
    Bivector<T> bivector{delta[0], delta[1], delta[2]};  // {e12, e13, e23}
    if (bivector.norm() > T(0.0)) {
      T theta = bivector.norm();
      Scalar<T> sin_theta{sin(theta)};
      Scalar<T> sinc_theta{sin(theta) / theta};
      Scalar<T> cos_theta{cos(theta)};
      B = B.unit();
      Vector<T> t{delta[3], delta[4], delta[5]};  // {e1ni, e2ni, e3ni}
      Vector<T> t_parallel = Op::project(t, B);
      Vector<T> t_perp = Op::reject(t, B);
      Vector<T> tt = cos_theta * t_perp + sinc_theta * t_parallel;
      auto ts = B * t_perp;
      motor_1 = Motor<T>{cos_theta[0], sin_theta[0] * bivector[0],
                         sin_theta[0] * bivector[1], sin_theta[0] * bivector[2],
                         tt[0], tt[1], tt[2], sin_theta[0] * ts[3]};
    } else {
      motor_1 = Motor<T>{T(1.0),   delta[0], delta[1], delta[2],
                         delta[3], delta[4], delta[5], T(0.0)};
    }
    Motor<T> motor_0{x[0], x[1], x[2], x[3], x[4], x[5], x[6], x[7]};
    Motor<T> motor_2 = motor_1 * motor_0;
    for (int i = 0; i < 8; ++i) x_plus_delta[i] = motor_2[i];
    return true;
  }
};
```

$\theta = \pi/3$. The benchmarks were implemented using the Benchmark[3] library of Google. The geometric algebra implementations used were the Versor library with the implementation as presented in Listing 1, the hep-ga library using the implementation as presented in Listing 4 and the matrix based approach implemented using the Eigen matrix library with the implementation presented in Listing 2. The performance of the different geometric algebra implementations in combination with the Ceres and Adept automatic differentiation libraries is shown in Table 4.

All three libraries were able to compute the correct function and derivative values, however, there were significant performance differences. The best performing implementation was the combination of the Versor library with the forward mode implementation from Ceres with a mean computation time of $\mu = 528$ ns and a standard deviation of $\sigma = 7$ ns. The worst performing implementation was the combination of the Matrix implementation with the forward mode implementation in the Adept library with a mean computation time of $\mu = 48125$ ns and a standard deviation of $\sigma = 1504$ ns. Using the Adept library, the hep-ga implementation were faster than the Versor implementation. Using the Versor library, the number of statements to compute the derivative were 134 with a total of 170 operations. Using the hep-ga library, the number of statements were 95 with a total of 171 operations. The number of statements and operations using the 4×4 matrix approach

[3] https://github.com/google/benchmark.

TABLE 4. Experimental results of automatic differentiation of the multivector valued functions in (4.2)–(4.8)

AD Lib.	AD Type	GA Lib.	Ops.	Stmts.	Time μ [ns]	Time σ [ns]
Adept	Forward	Matrix	2077	1600	48125	1504
Adept	Reverse	Matrix	2077	1600	44373	557
Ceres	Forward	Matrix	N/A	N/A	6880	87
Adept	Forward	Versor	170	134	5453	326
Adept	Reverse	Versor	170	134	5410	67
Ceres	Forward	Versor	N/A	N/A	528	7
Adept	Forward	hep-ga	171	95	3650	59
Adept	Reverse	hep-ga	171	95	3695	115
Ceres	Forward	hep-ga	N/A	N/A	681	7

The best performance was with the Versor geometric algebra library in combination with the forward mode automatic differentiation implementation in Ceres

differs in an order of magnitude to the number of statements and operations using the Versor and hep-ga library with a total of 1600 statements with 2077 operations. The reason for this difference is the use of expression templates in both Eigen and Adept and that the compiler is not able to reduce the generated expressions. This is a known limitation of the Adept library as presented in the Adept documentation[4]. Note that the hep-ga library is also based on expression templates. The matrix implementation were slowest in both automatic differentiation implementations. There were no significant differences using forward mode or reverse mode with the Adept library for these experiments as the size of the Jacobian were relatively small.

7.2. Multivector Estimation

This section presents experimental results from multivector estimation. Section 7.2.1 presents attitude estimation, that is, rotor estimation from unit (direction) vectors and Sect. 7.2.2 presents motor estimation from point clouds.

7.2.1. Rotor Estimation. Following the experimental setup in [39], the true vectors $\{a_i\}$ where $a_i \in \mathbb{R}^1_3$ for $i \in \{1, \dots, N\}$ are Gaussian distributed unit vectors with a standard deviation $\sigma = 0.8$. The vectors $\{a_i\}$ are rotated by the ground truth rotor R to form the set $\{a'_i\}$ where $a'_i = Ra_i\widetilde{R}$. Gaussian noise with standard deviation σ_r is added to the rotated vectors a'_i. The resulting vectors are then normalized to form the set $\{b_i\}$.

The cost function in (5.3) and the parameterization in (5.7) used in these experiments are implemented using the hep-ga library. Plots of the development in the cost function value in two experiments are shown in Fig. 1. The Levenberg-Marquardt method is able to estimate the rotor in 4 iterations, while the limited memory BFGS (L-BFGS) [36] method is able to estimate the rotor in 10 iterations.

[4] http://www.met.reading.ac.uk/clouds/adept/adept_documentation.pdf, p. 24.

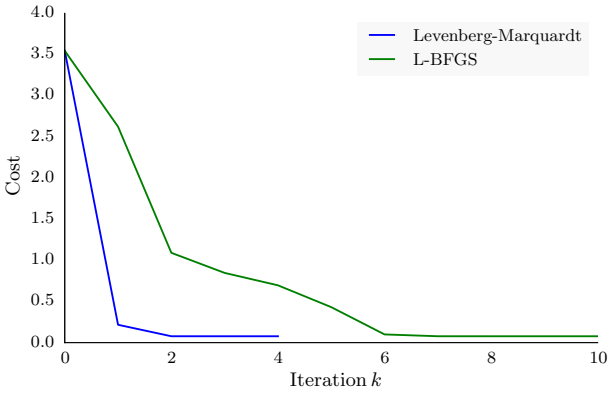

FIGURE 1. Plot of the value of the cost function in each iteration of two rotor estimation experiments using 10 observations. The rotor to be found is $R = \cos(\pi/6) - \sin(\pi/6)e_{12}$. The Gaussian noise that was added to form the set $\{b_i\}$ had a standard deviation $\sigma = 0.09$. The initial rotor was set to the identity rotor $R_0 = 1$. The bivector exponential parameterization was used in both experiments. The Levenberg–Marquardt method was able to estimate the rotor in 4 iterations while the limited memory BFGS (L-BFGS) method was able to estimate the rotor in 10 iterations

To compare the accuracy of our developed method with state of the art rotation estimation methods we compute the average mean and standard deviation of the root mean square (RMS) of the quality measure

$$\theta_i = \arccos(a_i \cdot \underline{a}_i) \tag{7.1}$$

from 1000 experiments, where $\{\underline{a}_i\}$ are the ground truth vector rotated by the estimated rotor $\underline{R}$.

Our method is denoted rotor, the quaternion method shipped with Ceres [1] is denoted ceres-quaternion and the singular value decomposition based method presented in [47] is denoted umeyama. The results are presented in Table 5. Our rotor-method and the ceres-quaternion method perform equally. This is expected, as the only difference is that the ceres-quaternion method is implemented with the matrix representation of a quaternion, rather than the geometric algebra implementation in the rotor-method. Both nonlinear methods perform slightly better than the singular value decomposition based method.

7.2.2. Motor Estimation. The optimization method used in this section is the Levenberg-Marquardt [33] method in combination with a dense QR linear solver. The focus is on the choice of cost function, the influence of the motor parameterization used and on the computational performance. All cost functions and parameterizations are implemented using the Versor library in combination with the automatic differentiation implementation in Ceres.

TABLE 5. Experimental results from rotor estimation

	$\sigma_r = 0.09$		$\sigma_r = 0.18$	
	$\overline{\mu}$	$\overline{\sigma}$	$\overline{\mu}$	$\overline{\sigma}$
Rotor	0.1393	0.0624	0.2861	0.1187
Ceres-quaternion	0.1393	0.0624	0.2861	0.1187
Umeyama	0.1455	0.0648	0.3010	0.1649

TABLE 6. Experimental results from motor estimation

#	Cost Func.	Param.	Residuals	Threads	Iterations	Time (s)
1	(5.9)	(5.15)	1000	1	26	0.0574
2	(5.9)	(5.14)	1000	1	26	0.0616
3	(5.10)	(5.15)	1000	1	4	0.0135
4	(5.10)	(5.14)	1000	1	5	0.0174
5	(5.9)	(5.15)	100000	512	26	3.1323
6	(5.9)	(5.14)	100000	512	26	3.2734
7	(5.10)	(5.15)	300000	512	4	0.7400
8	(5.10)	(5.14)	300000	512	5	0.9868
9	(5.9)	(5.15)	1000000	2048	26	29.5933
10	(5.9)	(5.14)	1000000	2048	26	32.0214
11	(5.10)	(5.15)	3000000	2048	4	7.1343
12	(5.10)	(5.14)	3000000	2048	5	9.4692

Experimental results from motor estimation using 1000, 100000 and 1000000 observations are shown in Table 6.

Similarly to the experimental setup in the previous section, the true points $\{a_i\}$, $a_i \in \mathbb{R}^1_{4,1}$ for $i \in \{1, \dots, n\}$ are Gaussian distributed with a standard deviation $\sigma = 0.8$. The points $\{a_i\}$ are transformed by the ground truth motor M to form the set $\{a'_i\}$ where $a'_i = Ma_i\widetilde{M}$. Gaussian noise with standard deviation $\sigma_r = 0.09$ is added to the rotated points a'_i. The resulting points form the set $\{b_i\}$.

The main difference between the two motor estimation formulations in (5.9) and (5.10) is that the formulation in (5.9) converges more slowly than (5.10) when close to the solution. The formulation in (5.9) converged in 26 iterations and the formulation in (5.10) converged in 4 iterations. The main reason for this difference is that the inner product based formulation in (5.9) employs only a measure for the total residual distance opposed to signed residual distances along each of the three coordinate axes in (5.10). These results were the same for all number of observations.

The choice of parameterization also influences the robustness and convergence rate of the optimization. The difference between the two parameterizations in Sect. 5.2.1 is that the parameterization in (5.15), employing 6 parameters and the bivector exponential map, removes null directions in the update, that is, update directions normal to the motor manifold $\mathcal{M}$. In contrast to this, when the parameterization in (5.14) is used, the updates can

be in all directions of $\mathbb{M}$. The use of the parameterization in (5.15) reduces the number of iterations from 5 to 4 when using the cost function in (5.10) employing 3 residuals per observation. It does not influence the convergence rate of the optimization when using the inner product based cost function in (5.9) with 1 residual per observation.

Even though the choice of parameterization did not influence the number of iterations when using the cost function in (5.9), it does influence the total time of the optimization. This reduction is also seen when using the cost function in (5.10). The reason for the time reduction is the reduction in the size of the Jacobian matrix, e.g., from a 3000000×8 Jacobian matrix in experiment 12 to a 3000000×6 Jacobian matrix in experiment 11, as seen in the two bottom rows of Table 6. This leads to a reduction in the time used to evaluate the Jacobian matrix and in the linear solver in each iteration.

The developed framework can be used to compute motors from large sets of observations. In practice, these observations would come from, e.g., RGB-D sensors. When estimating motors from synthetic datasets of 100000 points we were able to compute the correct motor in 4 iterations in 0.74 s when the number of OpenMP [10] threads available were set to 512 and from synthetic datasets of 1000000 points we were able to compute the correct motor in 4 iterations in 7.1343 s when the number of OpenMP [10] threads available were set to 2048.

8. Conclusion

In this work, we have presented automatic differentiation of multivector valued functions using 3 different implementations of geometric algebra and 2 different implementations of automatic differentiation and thus enabling automatic computations of gradient and Jacobian matrices for use in nonlinear least squares optimization. We have shown that our formulations perform equally well as state-of-the-art formulations for rotation estimation and we were able to estimate Euclidean rotors from noisy data in 4 iterations using the Levenberg–Marquardt optimization method. We have also presented motor estimation, that is, estimation of rigid body motions from noisy point data. We were able to estimate motors from 100000 point observations in 0.74 s and from 1000000 point observations in 7.1343 s, showing that our formulations scale well.

The presented formulations for multivector estimation can easily be expanded from estimation of rotors and motors using vectors and points to estimation of any transformation or object in the conformal model. The cost function can be arbitrary complex and difficult to differentiate analytically as all gradients and Jacobian matrices are computed using automatic differentiation.

Acknowledgments

Lars Tingelstad would like to thank Pablo Colapinto for the discussions at the AGACSE 2015 conference and acknowledge his work in developing the Versor geometric algebra library.

References

[1] Agarwal, S., Mierle, K. et al.: Ceres Solver. http://ceres-solver.org

[2] Agarwal, S., Snavely, N., Seitz, S.M., Szeliski, R.: Bundle Adjustment in the Large. In: Daniilidis, K., Maragos, P., Paragios, N. (eds.) ECCV (2), vol. 6312, Lecture Notes in Computer Science, pp. 29–42. Springer, Berlin (2010)

[3] Baydin, A.G., Barak, A.: Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. CoRR (2015). arXiv:abs/1502.05767

[4] Bayro-Corrochano, E., Daniilidis, K., Sommer, G.: Motor algebra for 3D kinematics: the case of the hand-eye calibration. J. Math. Imaging Vis. **13**(2), 79–100 (2000)

[5] Cibura, C.: Fitting Model Parameters in Conformal Geometric Algebra to Euclidean Observation Data. PhD thesis, University of Amsterdam (2012)

[6] Clifford, W.K.: Preliminary Sketch of Biquaternions. Proc. Lond. Math. Soc. **4**, 361–395 (1873)

[7] Colapinto, P.: Versor: Spatial Computing with Conformal Geometric Algebra. Master's thesis, University of California at Santa Barbara (2011). http://versor.mat.ucsb.edu

[8] Colapinto, P.: Boosted surfaces: synthesis of meshes using point pair generators as curvature operators in the 3d conformal model. Adv. Appl. Clifford Algebras **24**(1), 71–88 (2014)

[9] Colapinto, P.: Composing Surfaces with Conformal Rotors. In: Advances in Applied Clifford Algebras. pp. 1–22 (2016)

[10] Dagum, L., Menon, R.: OpenMP: an industry standard API for shared-memory programming. Comput. Sci. Eng. IEEE. **5**(1), 46–55 (1998)

[11] Davis, T.: SuiteSparse. http://faculty.cse.tamu.edu/davis/suitesparse.html. Online Accessed 25 July 2016

[12] Dorst, L.: The Representation of Rigid Body Motions in the Conformal Model of Geometric Algebra. In: Rosenhahn, B., Klette, R., Metaxas, D. (eds.) Human Motion, vol. 36, Computational Imaging and Vision, pp. 507–529. Springer (2008)

[13] Dorst, L.: Conformal Geometric Algebra by Extended Vahlen Matrices. Methods of Information in Medicine, 1 (2009)

[14] Dorst, L., Fontijne, D., Mann, S.: Geometric Algebra for Computer Science: An Object-Oriented Approach to Geometry. Morgan Kaufmann Publishers Inc., San Francisco (2007)

[15] Dorst, L., Fontijne, D., Mann, S.: Gaviewer. http://www.geometricalgebra.net/gaviewer_download.html (2010). Online Accessed 25 July 2016

[16] Fontijne, D.: Gaigen 2.5. http://sourceforge.net/projects/g25/. Online Accessed 25 July 2016

[17] Fontijne, D.: Efficient Implementation of Geometric Algebra. PhD thesis, University of Amsterdam (2007)

[18] Fontijne, D., Dorst, L., Bouma, T.: Gaigen. https://sourceforge.net/projects/gaigen/. Online Accessed 07 July 2016

[19] Fowler, M.: Domain Specific Languages, 1st edn. Addison-Wesley Professional(2010)

[20] Gebken, C.: Conformal Geometric Algebra in Stochastic Optimization Problems of 3D-Vision Applications. PhD thesis, University of Kiel (2009)

[21] Gebken, C., Perwass, C., Sommer, G.: Parameter estimation from uncertain data in geometric algebra. Adv. Appl. Clifford Algebras **18**(3-4), 647–664 (2008)

[22] Griewank, A., Walther, A.: Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation, 2 edn. SIAM, Philadelphia (2008)

[23] Guennebaud, G., Jacob, B. et al.: Eigen v3. http://eigen.tuxfamily.org (2010). Online; Accessed 25 July 2016

[24] Hartley, R., Zisserman, A.: Multiple View Geometry in Computer Vision, 2 edn. Cambridge University Press, New York (2003)

[25] Hestenes, D.: New Foundations for Classical Mechanics, vol. 15, Fundamental Theories of Physics. Springer (1986)

[26] Hestenes, D., Sobczyk, G.: Clifford Algebra to Geometric Calculus—A Unified Language for Mathematics and Physics, vol. 5, Fundamental Theories of Physics. Springer (1984)

[27] Hildenbrand, D., Pitt, J., Koch, A.: Gaalop—High Performance Parallel Computing Based on Conformal Geometric Algebra. In: Bayro-Corrochano, E., Scheuermann, G. (eds.) Geometric Algebra Computing, pp. 477–494. Springer, Berlin (2010)

[28] Hoffmann, P.H.W.: A Hitchhiker's Guide to Automatic Differentiation. In: Numerical Algorithms. pp. 1–37 (2015)

[29] Hogan, R.J.: Fast reverse-mode automatic differentiation using expression templates in C++. ACM Trans. Math. Softw. **40**(4), 26:1–26:16 (2014)

[30] Iglberger, K., Hager, G., Treibig, J., de Ulrich, R.: Expression templates revisited:a performance analysis of current methodologies. SIAM J. Sci. Comput. **34**(2), C42–C69 (2012)

[31] Lasenby, J., Fitzgerald, W.J., Lasenby, A.N., Doran, C.J.L.: New geometric methods for computer vision: an application to structure and motion estimation. Int. J. Comput. Vis. **26**(3), 191–213 (1998)

[32] Li, H., Hestenes, D., Rockwood, A.: Generalized Homogeneous Coordinates for Computational Geometry. In: Sommer, G. (ed.) Geometric Computing with Clifford Algebras, pp. 27–59. Springer, Berlin (2001)

[33] Madsen, K., Nielsen, H.B., Tingleff, O.: Methods for Non-Linear Least Squares Problems (2004)

[34] Munaro, M., Basso, F., Menegatti, E.: OpenPTrack: Open source multi-camera calibration and people tracking for RGB-D camera networks. Robot. Auton. Syst. **75**(Part B), 525–538 (2016)

[35] Nickolls, J., Buck, I., Garland, M., Skadron, K.: Scalable Parallel Programming with CUDA. Queue. **6**(2), 40–53 (2008)

[36] Nocedal, J., Wright, S.J.: Numerical Optimization, 2 edn. Springer, New York (2006)

[37] Perwass, C.: Applications of Geometric Algebra in Computer Vision—The geometry of multiple view tensors and 3D-reconstruction. PhD thesis, University of Cambridge (2000)

[38] Perwass, C.: CLUCalc Interactive Visualization. http://www.clucalc.info/ (2010). Online; Accessed 25 July 2016

[39] Perwass, C., Gebken, C., Sommer, G.: Estimation of Geometric Entities and Operators from Uncertain Data. In: Pattern Recognition, vol. 3663, Lecture Notes in Computer Science, pp. 459–467. Springer, Berlin (2005)

[40] Schmidt, J., Niemann, H.: Using Quaternions for Parametrizing 3-D Rotations in Unconstrained Nonlinear Optimization. In: VMV, pp. 399–406. Aka GmbH (2001)

[41] Schwan, C.: hep-ga: An Efficient Numeric Template Library for Geometric Algebra

[42] Snygg, J.: Clifford Algebra in Euclidean 3-Space. In: A New Approach to Differential Geometry using Clifford's Geometric Algebra, pp. 3–25. Birkhäuser, Boston (2012)

[43] Sobczyk, G.: Geometric matrix algebra. Linear Algebra Appl. **429**(5–6), 1163–1173 (2008)

[44] Sommer, H., Pradalier, C., Furgale, P.: Automatic Differentiation on Differentiable Manifolds as a Tool for Robotics. In: Int. Symp. on Robotics Research (ISRR) (2013)

[45] Stone, J.E., Gohara, D., Shi, G.: OpenCL: a parallel programming standard for heterogeneous computing systems. IEEE Des. Test. **12**(3), 66–73 (2010)

[46] Tingelstad, L.: GAME—Geometric Algebra Multivector Estimation. http://github.com/tingelst/game/ (2016)

[47] Umeyama, S.: Least-squares estimation of transformation parameters between two point patterns. IEEE Trans. Pattern Anal. Mach. Intell. **13**(4), 376–380 (1991)

[48] Valkenburg, R., Alwesh, N.: Calibration of Target Positions Using Conformal Geometric Algebra. In: Dorst, L., Lasenby, J., (eds.) Guide to Geometric Algebra in Practice, pp. 127–148. Springer, London (2011)

[49] Valkenburg, R., Dorst, L.: Estimating Motors from a Variety of Geometric Data in 3D Conformal Geometric Algebra. In: Dorst, L., Lasenby, J. (eds.) Guide to Geometric Algebra in Practice, pp. 25–45. Springer, London (2011)

[50] Wareham, R., Cameron, J., Lasenby, J.: Applications of Conformal Geometric Algebra in Computer Vision and Graphics. In: Li, H., Olver, P.J., Sommer, G. (eds.) Computer Algebra and Geometric Algebra with Applications, pp. 329–349. Springer, Berlin (2005)

[51] Woo, M., Neider, J., Davis, T., Shreiner, D.: OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 1.2, 3rd edn. Addison-Wesley Longman Publishing Co., Inc., Boston (1999)

Lars Tingelstad and Olav Egeland
Department of Production and Quality Engineering
Faculty of Engineering Science and Technology
NTNU, Norwegian University of Science and Technology
Trondheim, Norway
e-mail: lars.tingelstad@ntnu.no

Olav Egeland
e-mail: olav.egeland@ntnu.no

Received: January 31, 2016.
Accepted: August 18, 2016.